solidity中多线继承-super关键字的线性化调用方式

---

title: solidity中多线继承-super关键字的线性化调用方式

date: 2022

tags:

---

**# ———————————————————————————**

**## 在一些博客或者文档中我们可能看到对solidity中super关键字的描述是“用于访问（最近的）父合约的一些函数或者状态变量”，这里有一个简单的具象化但不全面的例子：**

```

//定义一个基类合约

contract A{

string public str01 = "我是A";

function foo() public view virtual returns(string memory){

return str01;

}

}

//A基类合约的子合约（派生合约）

contract B is A{

string public str01 = "我是B";

function foo() public view override returns(string memory){

super.foo();

}

}

```

**#### 编译部署B合约，当我们在用外部调用的时候B.foo()函数的时候，但从关键字的理解上来说，super代表最近的父合约，那么这里的父合约就只有A合约，所以会调用A中的foo()方法；**

**## 但是当遇到稍微复杂一些的多线继承的时候（多线继承推荐看登链社区的solidity中文的官方文档``https://learnblockchain.cn/docs/solidity/contracts.html#index-17``），比如类似的钻石问题，我们就需要搞清楚这个时候的合约编译以及运行情况，这样才能更好的理清思路，这里仍然有一个简单的具象化的例子：**

```

// SPDX-License-Identifier: MIT

pragma solidity ^0.8.13;

/\* 继承树：

God

/ \

Adam Eve

\ /

people

\*/

contract God {

event Log(string message);

function foo() public virtual {

emit Log("God.foo called");

}

function bar() public virtual {

emit Log("God.bar called");

}

}

contract Adam is God {

function foo() public virtual override {

emit Log("Adam.foo called");

Adam.foo();

}

function bar() public virtual override {

emit Log("Adam.bar called");

super.bar();

}

}

contract Eve is God {

function foo() public virtual override {

emit Log("Eve.foo called");

Eve.foo();

}

function bar() public virtual override {

emit Log("Eve.bar called");

super.bar();

}

}

contract people is Adam, Eve {

function foo() public override(Adam, Eve) {

super.foo();

}

function bar() public override(Adam, Eve) {

super.bar();

}

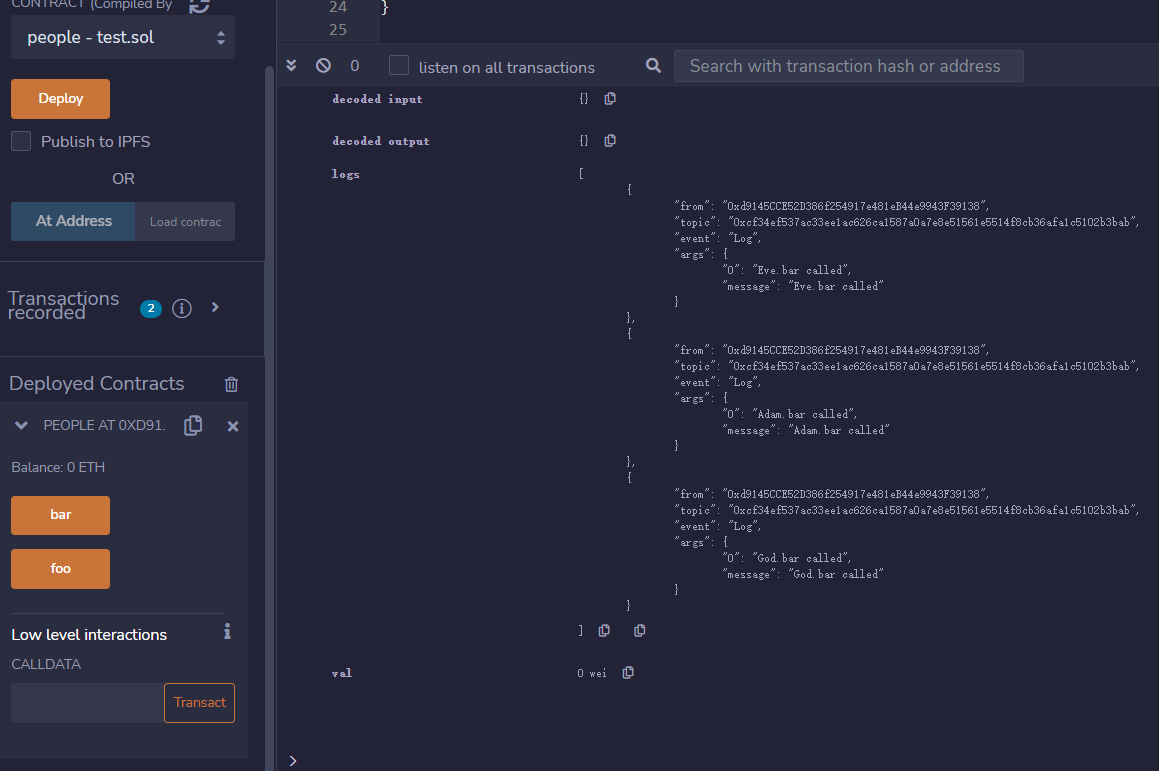
}

//代码来自：https://github.com/AmazingAng/WTF-Solidity/blob/main/13\_Inheritance/readme.md

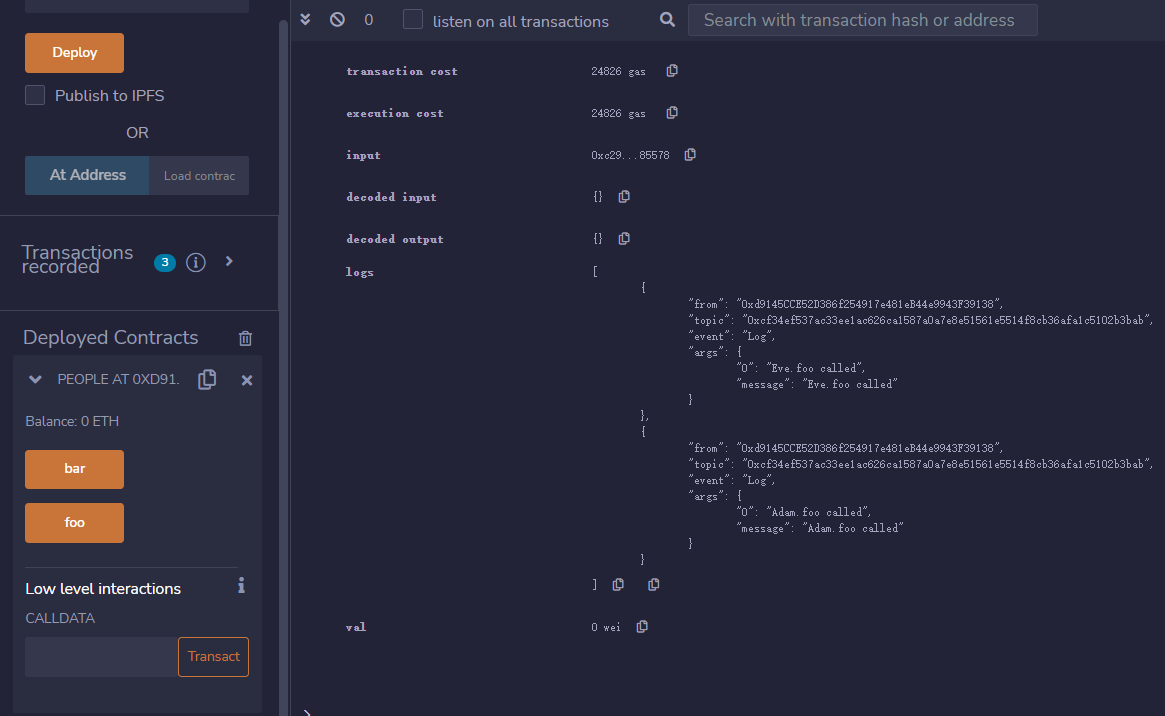
```

**## 先分析一下这个钻石问题的多线继承关系和合约内容，God合约做为基类合约有两个待重写的方法--bar()和foo(),当God合约的函数被调用时会触发God的Log事件；Adam和Eve做为God的派生合约都继承了God合约并且都重写了待重写方法-foo()和bar方法，同时，调用时会触发各自的事件peole继承foo()和bar()合约，作为其子类重写两个方法**

**#### 上面合约编译后部署people合约，调用bar()函数，出现结果：**



**#### 上面合约编译后部署people合约，调用foo()函数，出现结果：**



**## 如果你觉得这个运行结果在你的预期之内是一模一样的，那么说明你已经考虑到了：**

**``1.多线继承时合约的编译顺序是如何的``;**

**``2.super关键字的“最近”是紧紧跟上编译顺序的，而不是简单的父合约``；**

**总之，这里强调一个问题，在多线继承的时候一定要注意继承时合约的编译顺序（与python相反，一些文章中可以读到 这里不详细了解），并且进一步需要理解的是solidity中合约的编译顺序不是EVM内部决定的，而是依靠我们书写的顺序人为安排的。**

**## 那么带着这个结论和对函数调用出现的结果的疑问，先看调用bar函数（依照步骤理解）：**

**##### 1.明白编译顺序在编译people合约的时候，people合约继承了两个合约（Aam和Eve），这两个合约都是God的派生合约，那么，编译顺序就是从基类向下：God-->Adam-->Eve-->people(最后是本合约)，或者我们可以改变Eve和Adam的书写顺序``contract people is Eve,Adam``，那么编译顺序就变成了God->Eve-->God-->people**

**##### 2.调用bar()函数：执行super.bar，此时依照编译顺序，“最近的父合约”应该是Eve,所以执行Eve.bar()函数**

**##### 3.Eve.bar()函数首先触发事件，然后调用super.bar()，这时依照编译顺序，“最近的父合约”变为Adam，所以执行Adam.bar()函数**

**##### 4.Adam,bar()函数首先触发事件，然后调用super.bar(),这时依照编译顺序，“最近的父合约”变为基类合约--God合约，执行God.bar()**

**##### 5.God.bar()触发事件**

**##### 最后，按次序的三个事件被触发**

**## 同样的依照多线继承的合约编译顺序和super关键字的“最近的父合约”的思路，调用foo()函数，根据代码同样可以知道运行结果是会先触发Eve事件，然后触发Adam事件。**

**# 总结：**

**##### 这样理解或许是比较的冗余，但是 不乏是为了理解原理和执行方式，并且在很多官方文档或者博客并没有对此说明，在没有过多的学过c++或者python的多继承时，希望这篇文章会对你有帮助！**